Problem M3.11: VLIW Programming [?? Hours]

Problem M3.11.A

To get 1 cycle per vector element performance, we need to use loop unrolling and software pipelining. The original loop is unrolled four times and software pipelined. Two registers (**F3** and **F7**) are used for saving partial sums, which are summed at the end.

At the start of the program n may be any value. By making successive checks and providing fix-up code, n can be guaranteed to be positive and a multiple of 4 by the prolog.

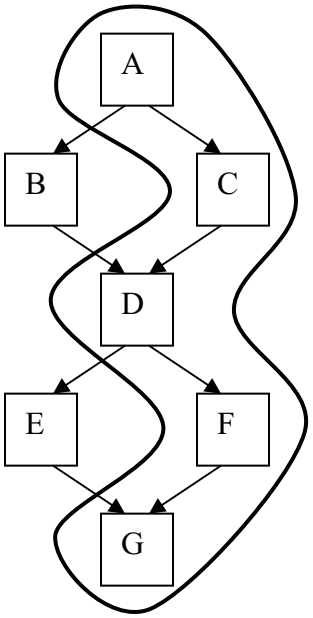
```
// R1 - points to X
// R2 - points to Y
// R5 - n
// F7 - result

    // clear partial sum registers
    MOVI2FP F3,R0
    MOVI2FP F7,R0

    // clear temporary registers used for multiply results
    MOVI2FP F2,R0
    MOVI2FP F6,R0
    MOVI2FP F10,R0
    MOVI2FP F14,R0

    // n must be greater than 0
    SGT     R3,R5,R0
    BEQZ    R3,end     // if !(n>0) goto end

    // n must be greater than 0
    ANDI  R3,R5,#3
    BEQZ  R3,prolog

    // (n>0) && ((n%4)!=0)
    SUB   R5,R5,R3
L1:
    L.S   F3,0(R1); L.S  F4,0(R2); SUBI R3,R3,#1
    MUL.S F3,F3,F4; ADDI R1,R1,#4;
    ADD.S F7,F7,F3; ADDI R2,R2,#4; BNEZ R3,L1

    BEQZ  R5,end

    // (n>=4) && ((n%4)==0)
prolog:
    L.S  F0, 0(R1); L.S  F1, 0(R2); SUBI R5,R5,#4
    L.S  F4, 4(R1); L.S  F5, 4(R2); ADDI R1,R1,#16
    L.S  F8,-8(R1); L.S  F9, 8(R2); ADDI R2,R2,#16
    L.S F12,-4(R1); L.S F13,-4(R2); BEQZ R5,epilog

    L.S  F0, 0(R1); L.S  F1, 0(R2); MUL.S  F2, F0, F1; SUBI R5,R5,#4
    L.S  F4, 4(R1); L.S  F5, 4(R2); MUL.S  F6, F4, F5; ADDI R1,R1,#16
    L.S  F8,-8(R1); L.S  F9, 8(R2); MUL.S F10, F8, F9; ADDI R2,R2,#16
    L.S F12,-4(R1); L.S F13,-4(R2); MUL.S F14,F12,F13; BEQZ R5,epilog

loop:
    L.S  F0, 0(R1); L.S  F1, 0(R2); MUL.S  F2, F0, F1; ADD.S F3,F3, F2; SUBI R5,R5,#4
```

```
    L.S  F4, 4(R1); L.S  F5, 4(R2); MUL.S  F6, F4, F5; ADD.S F7,F7, F6; ADDI R1,R1,#16
    L.S  F8,-8(R1); L.S  F9, 8(R2); MUL.S F10, F8, F9; ADD.S F3,F3,F10; ADDI R2,R2,#16
    L.S F12,-4(R1); L.S F13,-4(R2); MUL.S F14,F12,F13; ADD.S F7,F7,F14; BNEZ R5,loop

epilog:
    MUL.S  F2, F0, F1; ADD.S F3,F3, F2
    MUL.S  F6, F4, F5; ADD.S F7,F7, F6
    MUL.S F10, F8, F9; ADD.S F3,F3,F10
    MUL.S F14,F12,F13; ADD.S F7,F7,F14

    ADD.S F3,F3, F2
    ADD.S F7,F7, F6
    ADD.S F3,F3,F10
    ADD.S F7,F7,F14

    ADD.S F7,F7,F3

end:
```

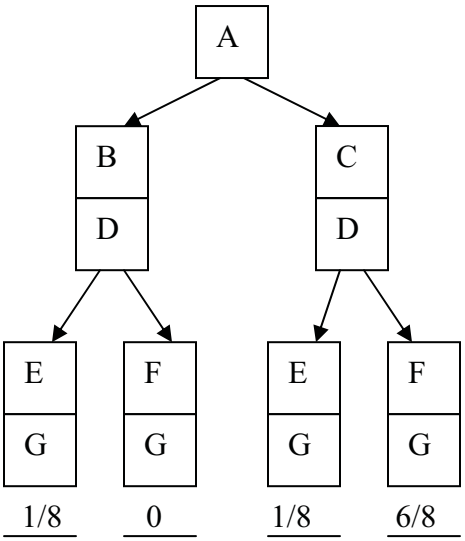
Problem M3.12: Trace Scheduling

Problem M3.12.A

Program's control flow graph



Decision tree



Problem M3.12.B

```
ACF:    ld   r1, data
        div  r3, r6, r7 ;; X <- V2/V3
        mul  r8, r6, r7 ;; Y <- V2*V3
D:      andi r2, r1, 3  ;; r2 <- r1%4
        bnez r2, G
A:      andi r2, r1, 7  ;; r2 <- r1%8
        bnez r2, E
B:      div  r3, r4, r5 ;; X <- V0/V1
E:      mul  r8, r4, r5 ;; Y <- V0*V1
G:
```

Problem M3.12.C

Assume that the load takes x cycles, divide takes y cycles, and multiply takes z cycles. Approximately how many cycles does the original code take? (ignore small constants)
x+max(y,z)

Approximately how many cycles does the new code take in the best case? **max(x,y,z)**

Problem M3.13: VLIW machines [?? Hours]

Problem M3.13.A

See **Table M3.13-1** on the next page.

Problem M3.13.B

12 cycles, 2/12=0.17 flops per cycle

Problem M3.13.C

3 instructions, because there are 5 memory ops and 5 ALU ops, and we can only issue 2 of them per instruction. (OR 4 instructions, because the slowest operation has a 4-cycle latency.)

Here is the resulting code.

add r1, r1, 4	add r2, r2, 4	ld f1, 0(r1)	ld f2, 0(r2)		fmul f4, f2, f1
add r3, r3, 4	add r4, r4, -1	ld f3, -4(r3)	st f4, -8(r1)	fadd f5, f4, f3	
	bnez r4, loop		st f5, -12(r3)		

for a particular instruction, white background corresponds to first iteration of the loop, grey background to the second iteration, yellow background to third, and blue to fourth. Note, one does not need to write the code to get an answer, because it's just a question of how many instructions are needed to express all the operations.

Problem M3.13.D

2/3=0.67 flops per cycle, 4 iterations at a time.

ALU1	ALU2	MU1	MU2	FADD	FMUL
add r1, r1, 4	add r2, r2, 4	ld f1, 0(r1)	ld f2, 0(r2)		
add r3, r3, 4	add r4, r4, -1	ld f3, 0(r3)			
					fmul f4, f2, f1
			st f4, -4(r1)	fadd f5, f4, f3	
	bnez r4, loop	st f5, -4(r3)			

Table M3.13-1: VLIW Program

Problem M3.13.E

We would need 5 instructions to execute two iterations and we would get 4/5=0.8 flops/cycle.

Problem M3.13.F

Same as above - 0.8 flops/cycle. We are fully utilizing the memory units, so we can't execute more loops/cycle.

Problem M3.13.G

No. We need to unroll the loop once to have an even number of memory ops. Use of the rotating registers would not allow us to squeeze in more memory ops per iteration, so we'd still need 5 instructions.

Problem M3.13.H

This is actually rather tricky. The correct answer is 5, because without interlocks, we can use the registers just as values come in for them, using the execution units to "store" the loops. The intuitive answer is 100 though.

Problem M3.13.I

There are approximately 100 instructions required, because maximum latency will be 100 cycles.

Problem M3.14: VLIW & Vector Coding [?? Hours]

Ben Bitdiddle has the following C loop, which takes the absolute value of elements within a vector.

```
for (i = 0; i < N; i++) {
    if (A[i] < 0)
        A[i] = -A[i];
}
```

Problem M3.14.A

```
; Initial Conditions:
;     R1 = N
;     R2 = &A[0]


      SGT R3, R1, R0
      BEQZ R3, end                                  ; R3 = (N > 0) | special case N ≤ 0
loop: LW R4, 0(R2)        | SUBI R1, R1, #1         ; R4 = A[i] | N--
      SLT R5, R4, R0      | ADDI R2, R2, #4         ; R5 = (A[i] < 0) | R2 = &A[i+1]
      BEQZ R5, next       |                         ; skip if (A[i]≥0)
      SUB R4, R0, R4      |                         ; A[i] = -A[i]
      SW R4, -4(R2)       |                         ; store updated value of A[i]
next: BNEZ R1, loop       |                         ; continue if N > 0
end:
```

Average Number of Cycles: ½ × (6 + 4) = 5

```
; SOLUTION #2

      SGT R3, R1, R0
      BNEZ R3, end                                  ; R3 = (N > 0) | special case N ≤ 0
loop: LW R4, 0(R2)        | SUBI R1, R1, #1         ; R4 = A[i] | N--
      SLT R5, R4, R0      | ADDI R2, R2, #4         ; R5 = (A[i] < 0) | R2 = &A[i+1]
      BNEZ R5, next       | SUB R4, R0, R4          ; skip if (A[i]≥0) | A[i] = -A[i]
      SW R4, -4(R2)       |                         ; store updated value of A[i]
next: BNEZ R1, loop       |                         ; continue if N > 0
end:
```

Average Number of Cycles: ½ × (5 + 4) = 4.5

NOTE: Although this solution minimizes code size and average number of cycles per element for this loop, it causes extra work because it subtracts regardless of whether it has to or not.

Problem M3.14.B

```
      SGT R3, R1, R0
      BNEZ R3, end                                  ; R3 = (N > 0) | if N ≤ 0
loop: LW R4, 0(R2)              | SUBI R1, R1, #1  ; R4 = A[i] | N--
      CMPLTZ P0, R4             | ADDI R2, R2, #4  ; P0 = (A[i]<0) | R2 = &A[i+1]
      (P0) SUB R4, R0, R4       |                  ; A[i] = -A[i]
      (P0) SW R4, -4(R2)        | BNEZ R1, loop    ; store updated value of A[i]
end:                                               ; continue if N > 0
```

Average Number of Cycles: ½ × (4 + 4) = 4 Cycles

Problem M3.14.C

```
; Initial Conditions:
;     R1 = N
;     R2 = &A[i]

R3 = N > 0
R4 = A[i]
R5 = N odd
R6 = A[i+1]

      SGT R3, R1, R0
      BEQZ R3, end              | ANDI R5, R1, #1
      BEQZ R5, loop             | LW R4, 0(R2)
      CMPLTZ P0, R4             | SUBI R1, R1, #1
      ADDI R2, R2, #4           | (P0) SUB R4, R0, R4
      (P0) SW R4, -4(R2)        | BEZ R1, end

loop: LW R4, 0(R2)              | SUBI R1, R1, #2
      CMPLTZ P0, R4             | LW R6, 4(R2)
      (P0) SUB R4, R0, R4       | CMPLTZ P1, R6
      (P0) SW R4, 0(R2)         | (P1) SUB R6 R0, R6
                                | (P1) SW R6 4(R2)
      ADDI R2, R2, #8           | BNEZ R1, loop
end:
```

Average Number of Cycles: 6 for 2 elements = 3 cycles per element

Problem M3.14.D

```
; Initial Conditions:
;     R1 = N
;     R2 = &A[i]

      L.D F0, #0
      MTC1 VLR R1             # operate on all N elements
      CVM
      LV V1, R2               # load A
      SLTVS.D V1, F0          # setup the mask vector
      SUBSV.D V1, F0, V1      # negate appropriate elements
      SV R2, V1               # store back changes
```

Average Number of Cycles: ≈ (N/2 + N/2) / N ≈ 1 cycle per element (assuming chaining)

Note: Because there is only one ALU per lane, only the load and the SLT (Set-Less-Than) can be chained together, while the subtract and the store can be chained together. Execution time (per element) of the other instructions is negligible when N is large.

Problem M3.14.E

```
; assume m = known vector length
; Initial Conditions:
;     R1 = N
;     R2 = &A[i]

      L.D F0, #0
      ANDI R3, R1, (m-1)      # get N%m - assume m is a power of 2
      MTC1 VLR R3             # operate on first N%m elements
      LV V1, R2               # load A
      SLTVS.D V1, F0          # setup the mask vector
      SUBSV.D V1, F0, V1      # negate appropriate elements
      SV R2, V1               # store back changes
      SUB R1, R1, R3          # decrease i by N%m (i is divisible by m now)
      SLLI R3, R3, #2         # (we're counting i down)
      ADDI R2, R2, R3         # advance A pointer
      BEQZ R1, end            # i == 0 -> done
      ADDI R3, R0, m
      MTC1 VLR R3             # operate on all elements

loop:
      CVM
      LV V1, R2               # load A
      SLTVS.D V1, F0          # setup the mask vector
      SUBSV.D V1, F0, V1      # negate appropriate elements
      SV R2, V1               # store back changes
      ADDI R2, R2, (m*4)      # advance A pointer
      SUBI R1, R1, m          # decrease i by m
      BNEZ R1, loop           # done?

end:
      CVM
```

Problem M3.15: Predication and VLIW [?? Hours]

Problem M3.15.A

```
      l.s   f1, 0(r1)      ; f1 = *r1
      seq.s r5, f10, f1    ; r5 = (f10==f1)
      cmpnez p1, r5        ; p1 = (r5!=0)
(p1)  add.s f2, f1, f11    ; if (p1) f2 = f1+f11
(!p1) add.s f2, f1, f12    ; if(!p1) f2 = f1+f12
      s.s   f2, 0(r2)      ; *r2 = f2
```

Problem M3.15.B

See the next page (**Table M3.15-2**).

Label	integer op	floating point add	memory op	branch
loop:			l.s f1,0(r1)	
			l.s f3,4(r1)	
	addi r1, r1, #8	cmpnez p1, f1		
		cmpnez p3, f3		
		(p1) add.s f2, f1, f1		
		(p3) add.s f4, f3, f3		
			(p1) s.s f2, -8(r1)	
			(p3) s.s f4, -4(r1)	bneq r1, r2, loop

Table M3.15-1

label	integer op	floating point add	memory op	branch
			l.s f1,0(r1)	
			l.s f3,4(r1)	
	addi r1, r1, #8	cmpnez p1, f1		
		cmpnez p3, f3		beq r1, r2, epilog
loop:		**(p1) add.s f2, f1, f1**	**l.s f1,0(r1)**	
		(p3) add.s f4, f3, f3	**l.s f3,4(r1)**	
	addi r1, r1, #8	**cmpnez p1, f1**	**(p1) s.s f2, -8(r1)**	
		cmpnez p3, f3	**(p3) s.s f4, -12(r1)**	bneq r1, r2,loop
epilog:		(p1) add.s f2, f1, f1		
		(p3) add.s f4, f3, f3		
			(p1) s.s f2, -8(r1)	
			(p3) s.s f2, -4(r1)	

Table M3.15-2